# The FairRoot framework



2012 J. Phys.: Conf. Ser. 396 022001

(http://iopscience.iop.org/1742-6596/396/2/022001)

Journal of Physics: Conference Series **396** (2012) 022001 doi:10.1088/1742-6596/396/2/022001

# The FairRoot framework

**M. Al-Turany[1], D. Bertini[1], R. Karabowicz[1], D. Kresan[1], P. Malzacher[1], T. Stockmanns[2], F. Uhlig[1]**

[1] GSI - Helmholtzzentrum fur Schwerionenforschung GmbH, Plankstrasse 1, 64291 Darmstadt, Germany
[2] FZJ, Forschungszentrum Jülich GmbH, Germany

E-mail: f.uhlig@gsi.de

**Abstract.** The FairRoot framework is an object oriented simulation, reconstruction and data analysis framework based on ROOT. It includes core services for detector simulation and offline analysis. The framework delivers base classes which enable the users to easily construct their experimental setup in a fast and convenient way. By using the Virtual Monte Carlo concept it is possible to perform the simulations using either Geant3 or Geant4 without changing the user code or the geometry description. Using and extending the task mechanism of ROOT it is possible to implement complex analysis tasks in a convenient way. Moreover, using the FairCuda interface of the framework it is possible to run some of these tasks also on GPU. Data IO, as well as parameter handling and data base connections are also handled by the framework. Since some of the experiments will not have an experimental setup with a conventional trigger system, the framework can handle also free flowing input streams of detector data. For this mode of operation the framework provides classes to create the needed time sorted input streams of detector data out of the event based simulation data . There are also tools to do radiation studies and to visualize the simulated data. A CMake-CDash based building and monitoring system is also part of the FairRoot services which helps to build and test the framework on many different platforms in an automatic way, including also Continuous Integration.

## 1. Introduction

The FairRoot [1] project started in 2003/2004 as a software framework for the CBM experiment [2] at GSI/FAIR [3, 4] with two people working on the topic. In 2005/2006 when the experiment [5] decided to use FairRoot (at this time called CbmRoot and used only by the CBM experiment) the code base was splitted into the base part called FairRoot and the experiment specific parts called CbmRoot and PandaRoot . With this splitting it was easily possible for other experiments to use FairRoot as base for their simulation, reconstruction and analysis. Meanwhile the following experiments or groups use FairRoot as base for their simulation and analysis: CBM, PANDA, R3B [6], ASYEOS [7] and the GEM subgroup of FOPI [8], all of them at GSI/FAIR. With the MPD project [9] at JINR in Dubna and the EIC project [10] there are now also groups outside of GSI/FAIR using the framework. Currently there are 5 developers working on FairRoot, where most of these people are dedicated to one of the experiments. With this setup, where people also work directly for the experiments, the experiments are strongly connected to FairRoot and the development process of the code. Beside this core team there are additional developers from the different experiments which contributed to major parts of the base framework.

The framework is heavily used in the different experiments for detector- and performance studies, to do radiation studies and to deliver input for the technical design reports.

## 2. Core framework

Figure 1 shows schematically the design of FairRoot. FairRoot uses many features of the ROOT [11] package, not all of them shown in the figure. The middle of the figure shows the part which is called FairRoot and is described in more detail below. The lower part indicates the different experiments which use FairRoot to implement their experimental setup and to do their simulation and reconstruction.

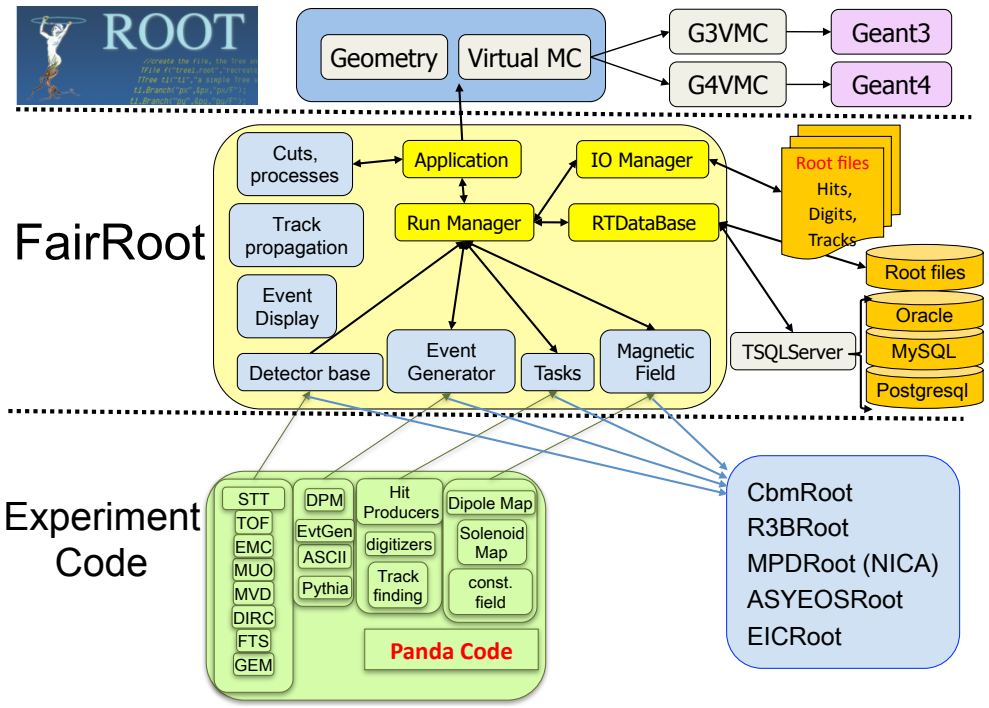


**Figure 1.** FairRoot Design

The FairRoot framework delivers on one hand managing classes which drive the execution of a simulation or reconstruction session, handle the IO to file, handle the parameter IO or are used to build the geometry for a simulation from different sorts of input files. On the other hand the framework delivers base classes for detector handling, magnetic field definition, event generators for simulation sessions as well as tasks which are used for the analysis sessions. Additionally it provides buffers and tasks which are needed for the time based simulation.

One of the design goals of FairRoot was to provide flexibility to the users, i.e. it should be possible to do for example simulations with different detector configurations or to test different tracking algorithms during the reconstruction without recompiling the code. To fulfill this requirements many settings must be configurable at run time, which means that the configuration has to be read at run time from input files. To avoid writing a parser for the configuration files ROOT CINT is used for this task, so the complete configuration to steer FairRoot is defined in ROOT macros.

FairRoot does not deliver an executable, but it is a set of shared libraries which are loaded on demand in a ROOT session, which means the root binary is the executable and the FairRoot libraries are plugged in. Moreover the user code is compiled to shared libraries and loaded on demand. This allows to load only the libraries which are necessary for the specific task, which reduces the amount of memory needed at run time. A typical size for a simulation session of the CBM or PANDA experiment is between 300MB and 500MB. The size for a reconstruction is slightly larger but stays well below 1GB.

Using the Virtual Monte Carlo (VMC) [12] it is possible to use the same user code for different Monte Carlo transport engines. At the moment Geant3 [13] and Geant4 [14] are supported. ROOT macros are also used to set the configuration of the used transport engines.

The main class of the framework is the *Run Manager* which handles the initialization, runs the event loop, controls the IO using the *IO manager* and the parameter handling by using the runtime database (*RTDataBase*). In simulation sessions it also sets up the needed environment and passes this information then to the VMC application (*Application*). The passed information includes material and geometry definition as well as information about magnetic field configurations, event generators and about the active and inactive detectors. In a simulation session the event loop is handled by the VMC Application and the Run Manager is only called when the execution reaches defined hooks. In the following we will describe some main parts of FairRoot in more detail.

### *2.1. Geometry interface*

The geometry interface is used to build the geometry from different input files in ROOT TGeoManager format, which is used by FairRoot internally. The geometries can be defined in an ASCII file of a special format inherited from the Hades experiment or in a ROOT file containing a TGeoManager or TGeoVolume geometry of the specific detector. There is also the possibility to put the geometry definitions using TGeo functionality explicitly in the code, which is not recommended since with this solution one loses the ability to change the geometry at run time without recompilation of the code. To be as flexible as possible the different detectors or modules (inactive parts of the experimental setup like the target or the beam pipe) have all separate geometry definitions, which are loaded from different input files. This makes changing the experimental setup as easy as changing the geometry definition name in the steering macro at run time. At run time all geometries for the different detectors are loaded, in case of ASCII input the TGeo geometry is constructed and then add to the main geometry manager. The different methods of geometry definitions as described above can be mixed, so one part of the geometry can be defined using the ASCII description while another part can be load from a ROOT file. At the end of the simulation session the complete geometry is saved in the parameter database. In reconstruction sessions the correct geometry is automatically read from the parameter database. To be able to run the different Monte Carlo transport codes also with their native geometry navigation system the TGeo geometry is converted to the native geometry model at run time using the VGM [15] package.

Using the TGeo geometry model has also the big advantage that one can use the ROOT event display with only minimal effort.

### *2.2. IO Manager*

The storage of information created either by the sensitive detectors in a simulation session or by the tasks in a reconstruction session is done on an event by event basis. The produced information is stored in data objects. For all the different data classes there are dedicated data objects, one for example which keeps all the data of a particle passing a specific sensitive detector. The data objects for a complete event which consists normally of many particles are stored in one ROOT TClonesArray. All the different data classes are organized in a ROOT TTree structure.

The TTree structure is dynamically created at run time depending on the experimental setup during the simulation or depending on the tasks in reconstruction sessions. The detectors in simulation or the tasks in reconstruction register the data objects they want to store at the IO manager. With this information the IO manager constructs the correct TTree structure for each session. The detectors or the tasks also define during the registration if the data should be transient or persistent. All the persistent objects are serialized and streamed into a ROOT file. The decision if the data is persistent or only transient is again made in the steering macro. As an example one can have a sequence of tasks which produce out of uncalibrated detector data in the first step some calibrated data and in a second step it combines the data to position information which can then be used for tracking. For debugging issues it is possible to write each data level to the output file, whereas for production sessions only the final data is written to file.

*2.3. Database handling*
Since the setup of a typical nuclear experiment is varying depending on the physics case under study, the software for simulation and analysis should be able to handle a large amount of different parameters and their variation in time. Most of this parameters will be present in different versions, for example due to changes in the calibration. This makes it necessary to have a parameter database with a well defined versioning system. For FairRoot one has now to distinguish between the *Runtime Database* which is an integral part of the framework and the *storage database* which is used to securely store the parameters. The concept of the parameter database is shown in fig. 2

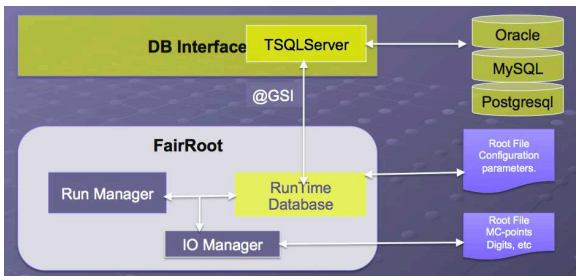


**Figure 2.** The Database interface in FairRoot.

*2.3.1. Runtime database* The Runtime database, based on the original Hades runtime database [16], is responsible to provide the correct parameters for the different tasks during a simulation or reconstruction session. The parameters are stored in parameter containers. For example one of these containers may contain all calibration parameters for one detector. The different tasks register the parameter containers they need in the runtime database, via the name of the parameter container. After the automatic initialization of the parameter containers the tasks can access the parameters. The runtime database manages the automatic initialization of the parameter containers from the storage and the saving of changed parameter containers if necessary. The containers can be initialized from up to two inputs where the possible inputs are ASCII files, ROOT files and databases. If more than one input is used the parameter containers which are not present in the first input are automatically taken from the second input. If a

parameter is changed during the session it will be saved to the defined output. At this time the version management is important. If the parameter containers are stored in ROOT files this is done by incrementing the stored object version. The next access of the object in the file will automatically load the newest version of the parameter container. An access to older versions is possible by defining the version number to be read. The version management for the storage database is described below.

*2.3.2. Connection to storage database* The FairRoot framework implements a database interface which provides a simple and uniform concept regardless of the data being accessed. Furthermore the database interface, using internally the ROOT TSQLServer services, can access data from more than one database instance using a dedicated database connectivity implementation. At initialization time, the order of a user-defined URLs list reflects the access priority. The first database in the list is used for searching the data, if it fails the next database in the list is used until the complete set of data can be retrieved. This gives the user the flexibility to create his own database from a subset of the official one and to put it ahead in the list.

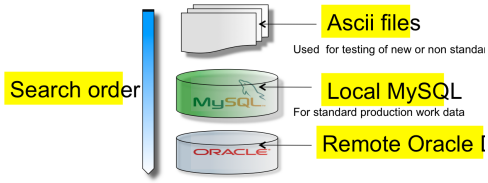


**Figure 3.** The ordered priority list of databases.

Ultimately, any of the data retrieved could depend on the run or the event being processed. Detector relevant parameters, such as calibration constants, will change with time and the interface has to retrieve the right ones for the current run or event. For this reason, all requests for data going through the interface must supply information about:

- The type of data: real or Monte Carlo simulation
- The type of Detector
- The date and time of the run or event (in UTC)

This information is called a Context. In the database all information is tagged by a Context i.e. by a validation range which identifies the type of data and detector as well as the ranges of date times for which it is valid.

Another important point is to minimize the IO. Some requests, particularly for detector relevant parameters, can pull in large amounts of data, however users must not load it once at the start of the job and then use it repeatedly since it may not be valid for all the data they process. Moreover multiple users may want access the same data and it would be inefficient for each of them to have their own copy. To deal with both of the above problems, the interface uses the concept of handle or proxy and implements different levels of objects caching.

## 3. Visualization Tools

Sometimes one needs the possibility to look at the experimental setup as it was used for a simulation, or one is interested if the created hits correspond with the underlying MC position information. For these tasks an event display (shown in fig. 4), based on the ROOT TEve package, is integrated in FairRoot. With this event display one can browse through the geometry and examine simulated or reconstructed events. It allows to set some filters, e.g to show only tracks of a defined particle type and in a defined energy range. Another important information when designing a detector or a complete experimental setup is the amount of material in the setup. For this task there is a special run mode in FairRoot, where all information about materials passed along a particle track are stored for later usage. An example of the usage of this information is shown in fig. 5 where the radiation length is shown as a function of the azimuthal angle.

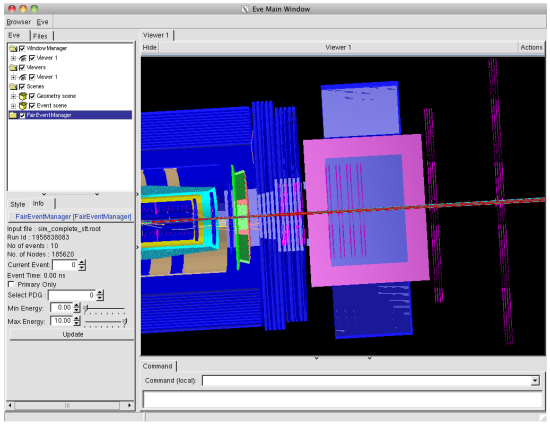

**Figure 4.** Event display of the PANDA experiment. Shown is a top view of the central and forward part.

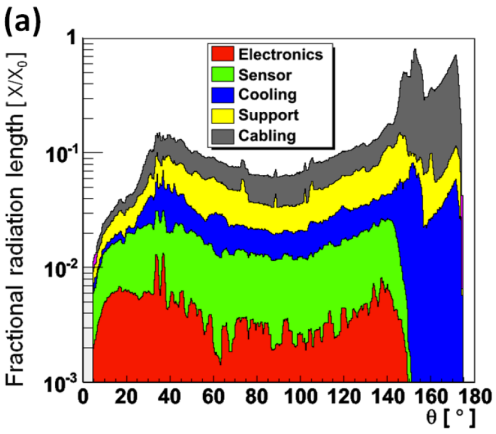


**Figure 5.** Summary of the radiation length as function of the angle for the MVD detector of the PANDA experiment. One can see that most of the material for cooling, support and cables are in the backward part of the setup where no detectors are located.

Beside the information about the material budget it is also important to have information about the accumulated dose in a given detector volume. For example the electronic properties of a semiconducting detector depend strongly on this accumulated dose which, at the end also defines the lifetime of the detector. To answer such questions there is also a special run mode, where for each volume of the setup (sensitive detector and passive materials) all the deposited energy together with additional information like for example particle type is saved. An example how this information is used is shown in fig. 6. Sometimes one is only interested in the number of particles which pass a certain boundary in the experiment. One solution would be to create

a sensitive dummy detector of the right dimensions and place it in the experimental setup. The drawbacks of this solution are the overhead one would introduce and more important it would not be possible to place the dummy detector at positions where already other volumes are located. To overcome this problems in FairRoot it is possible to define virtual surfaces which can be placed without interference with real volumes. The example in fig. 7 shows a virtual plane which cuts through the complete PANDA detectors at the target position.

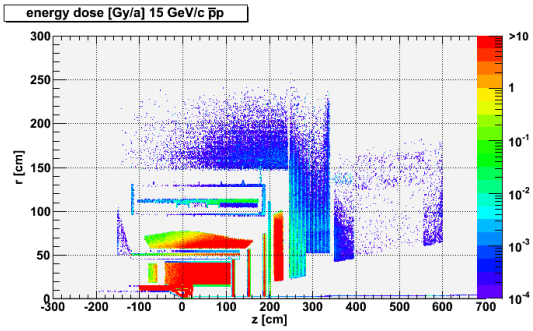


**Figure 6.** Accumulated dose for the different volumes of the PANDA spectrometer.

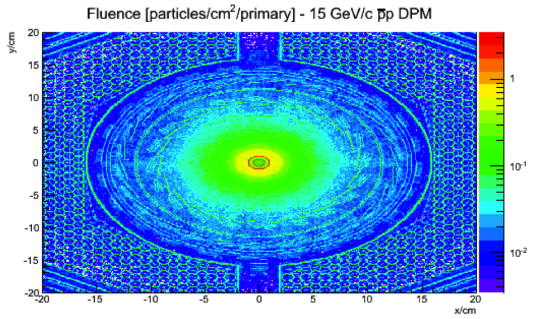


**Figure 7.** Particle fluency through a virtual plane at the target position of the PANDA experiment. In the inner part on can see the different layers of the Micro Vertex Detector. In the outer part one clearly sees the straw tubes of the Straw Tube Tracker.

## 4. Proof integration

The **P**arallel **ROO**T **F**acility (PROOF[17]) is a ROOT extension allowing analysis of large data sets in parallel on clusters of machines with multiple CPUs. The usage of the appliance requires the analysis to be dividable into independent subsets and the code to follow a definite organization (TSelector-derived class to manage task processing).

The recently implemented changes to FairRoot focus therefore on and around the FairAnaSelector class that inherits from ROOT::TSelector. The main goal, besides allowing user parallel data reconstruction, was to restrict the necessary code changes to the base classes thus reducing changes to detector classes and analysis steering macros. However already in the development and testing phase we found out that changes to the existing classes are unavoidable. The majority of them originate in the coders neglecting the coding rules, mainly not initializing member variables in the class constructor. After these fixes the only difference between running locally and on PROOF is adding one string "proof" in the analysis steering macro.

The FairRoot with PROOF analysis has been tested on several GSI machines and the reconstruction time improvement of up to 5 times was observed as compared to reconstruction performed on 1 CPU only. In this case a test PROOF cluster was set up using PoD[18] consisting of 3 machines with 4+8+8=20 CPUs altogether.

## 5. Time based simulations

The CBM and the PANDA experiment will operate at high event rates ($\approx 10^7 evt/s$) and do not foresee any hardware trigger in their Data Acquisition. Both experiments plan to use self triggered detectors, which stream the zero-suppressed relevant information to the next level of

the DAQ and finally into a compute farm for online event selection. All data objects send by the detectors consist of the data itself and a high quality time stamp.

In this setup an event is not any longer defined by the DAQ as in a classical triggered experiment, but has to be build by the online compute farm from the time-stamped data. Depending on the detector and the read out electronics the order of the signals reaching the DAQ will not follow the event order. The tasks for the DAQ and the online compute farm will be to build out of this data streams an event again, before a software trigger can be created.

To simulate such a scenario within the FairRoot framework first the output of the MC transport engines has to be transformed from an event based layout into the data streams. Out of this time-based data streams an event builder process has to put together events again which then can be analyzed by the already existing algorithms.

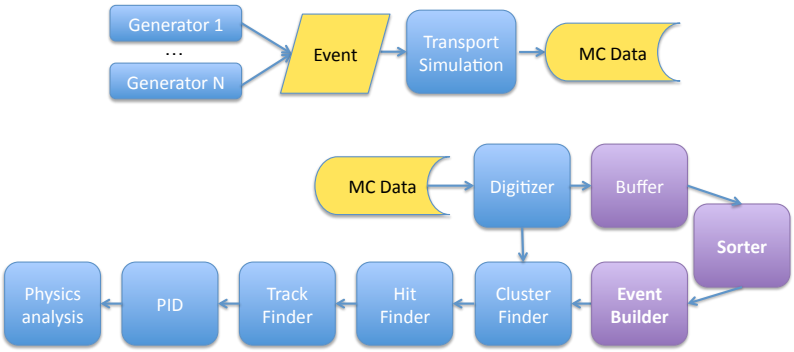


**Figure 8.** Sequence of tasks for a simulation session (upper part) and a reconstruction session (lower part).

For this tasks part of the structure of FairRoot had to be redesigned with the restriction that event based analysis must still be possible. The different steps to come from the input data of the MC simulation to physics results are shown in fig. 8. During the simulation session one or more generators fill information about particles to be transported by the MC simulation into an event stack. FairRoot already contains several generators, e.g. a generator which reads UrQMD data [19] from an input file. After particle generation (filling the event stack) the MC transport takes the particles one after the other from the stack and transports them through the experimental setup. At the end the output is written to file. The following steps in the reconstruction and analysis can be done either in one go, or it is also possible to save the data after each step and use this output as input for the next step. The first step of the reconstruction session is the conversion of perfect MC data into a data format similar to the one produced by the real detector (digitization). For each detector under study there is such a digitization task. The digitization is also the point where there is a splitting between time based and event based simulation. In an event based simulation the digitized data is written to the output container after each event. In case of a time based simulation the data is written first to a buffer which stores needed information between different events and implements functionality how to handle additional hits in the same detector channel. For each detector there is a maximum time defined until stored data can be changed. This "dead time" and also the functionality how to alter the buffered data is different for each detector, so this "detector response" has to be implemented according to the way the detector and the electronics work. As an example one detector may not recognize additional hits in the same detector channel until the electronics finish the read out process while for another detector which has a time over threshold electronics the time over threshold will change when there is an additional hit in the same detector channel.

For a time-based reconstruction the run manager defines for each event an event time which

is increasing with each event. This event time is calculated according to the user input and describe usually the expected beam properties. After each event the buffer checks which part of the stored data can be written to the output file. All the data where the actual event time is larger than the time when the signal happens plus the "dead time" of the detector can safely be written to the output file. All following events will have an even larger time and can not modify the data. The output of this step is a data stream which is not ordered in time. The next step is then to sort this data stream according to the time stamps. There is a base class for this task which implements a simple ring buffer and has to be customized for each detector by defining the number of storage cells and the time width of one storage cell. Both numbers are defined by the properties of the detector and the electronics. The last step to generate an event again is the so called event builder. This task has to read all the data which belongs to one event and write this data in an output container. The actual implementation depends very much on the detector, the experiment and also the event rate under consideration. Also for this task there is a base class which already implements different algorithms. The following algorithms are already available.

- Read all data which are still in the Array until a given absolute time is reached
- Read all data in a given time window
- Read all data until a time gap between two data elements of a given size is reached.

The output of the event builder task is again event by event, so the next task (as shown in fig. 8) isn't different for time based or event based reconstructions.

## 6. Conclusion

FairRoot is a framework for simulation, reconstruction and analysis, which is used by several large experimental collaborations. A brief introduction to the main features of the framework has been given. A special focus was given to the new challenge of simulating a free running triggerless data acquisition. The differences between such a time based and the classical event based simulation has been discussed.

[1] FairRoot core team, FairRoot, Website, available online at `http://fairroot.gsi.de`; last visited on May 16th 2012.
[2] CBM Collaboration, Compressed Baryonic Matter (CBM) experiment, Website, available online at `http://www.gsi.de/forschung/fair_experiments/CBM/index_e.html`; last visited on May 16th 2012.
[3] GSI GmbH, GSI Helmholtzzentrum fr Schwerionenforschung GmbH, Website, available online at `http://www.gsi.de/portrait/index_e.html`; last visited on May 16th 2012.
[4] Fair GmbH, FAIR - Facility for Antiproton and Ion Research in Europe GmbH, Website, available online at `http://www.fair-center.eu`; last visited on May 16th 2012.
[5] Panda Collaboration, PANDA experiment, Website, available online at `http://www-panda.gsi.de/`; last visited on May 16th 2012.
[6] R3B Collaboration, R3B experiment, Website, available online at `http://www.gsi.de/forschung/kp/kr/R3B_e.html`; last visited on May 16th 2012.
[7] R C Lemmon, P Russotto and the ASY-EOS collaboration, Constraining the nuclear symmetry energy at supra saturation densities GSI-Exp Proposal
[8] Bohmer F, Angerer H, Dorheim S, Hoppner C, Ketzer B *et al.* 2011 *Nucl.Phys.Proc.Suppl.* **215** 278–280
[9] MpdRoot software team, Simulation and Analysis Framework for NICA/MPD Detectors Webpage, available online at `http://mpd.jinr.ru/dr/`; last visited on June 22th 2012.
[10] EIC Collaboration, Letter of Intent for Detector R+D, Towards an EIC Detector, Webpage, available online at `https://wiki.bnl.gov/conferences/images/6/69/RD_2011-2_EICTrackingPIDproposal.pdf`; last visited on June 22th 2012.

[11] The Root Team, ROOT Homepage, Website, available online at `http://root.cern.ch/drupal/`; last visited on June 18th 2012.
[12] Ivana Hrivnacova et al, The Virtual Monte Carlo Presented at CHEP 2003, La Jolla, PSN THJT006., available online at `http://arxiv.org/pdf/cs/0306005v1`; last visited on June 22th 2012.
[13] Application Software Group, CN Div, GEANT Detector Description and Simulation Tool (Ver- sion 3.21) CERN Program Library W5013, available online at `http://wwwinfo.cern.ch/asd/geant`; last visited on June 22th 2012.
[14] Allison J, Amako K, Apostolakis J, Araujo H, Dubois P, Asai M, Barrand G, Capra R, Chauvie S, Chytracek R, Cirrone G, Cooperman G, Cosmo G, Cuttone G, Daquino G, Donszelmann M, Dressel M, Folger G, Foppiano F, Generowicz J, Grichine V, Guatelli S, Gumplinger P, Heikkinen A, Hrivnacova I, Howard A, Incerti S, Ivanchenko V, Johnson T, Jones F, Koi T, Kokoulin R, Kossov M, Kurashige H, Lara V, Larsson S, Lei F, Link O, Longo F, Maire M, Mantero A, Mascialino B, McLaren I, Lorenzo P, Minamimoto K, Murakami K, Nieminen P, Pandola L, Parlati S, Peralta L, Perl J, Pfeiffer A, Pia M, Ribon A, Rodrigues P, Russo G, Sadilov S, Santin G, Sasaki T, Smith D, Starkov N, Tanaka S, Tcherniaev E, Tome B, Trindade A, Truscott P, Urban L, Verderi M, Walkden A, Wellisch J, Williams D, Wright D and Yoshida H 2006 *Nuclear Science, IEEE Transactions on* **53** 270 –278 ISSN 0018-9499
[15] Ivana Hrivnacova, Virtual Geometry Model Homepage, Website, available online at `http://ivana.home.cern.ch/ivana/VGM.html`; last visited on June 22th 2012.
[16] Ilse König, Initialization in the HADES analysis program, Website, available online at `http://web-docs.gsi.de/ ilse/initialization.htm`; last visited on June 18th 2012.
[17] The Root Team, PROOF Homepage, Website, available online at `http://root.cern.ch/drupal/content/proof`; last visited on June 18th 2012.
[18] Anar Manafov, Proof on Demand Homepage, Website, available online at `http://pod.gsi.de`; last visited on June 18th 2012.
[19] Bass S, Belkacem M, Bleicher M, Brandstetter M, Bravina L *et al.* 1998 *Prog.Part.Nucl.Phys.* **41** 255–369 (*Preprint* `nucl-th/9803035`)